High-Contrast Algorithm Behavior
Observation, Hypothesis, and Experimental Design

Matthias F. Stallmann and Franc Bringlez

Observation in a limited domain
- Initial experiments: design
- Initial experiments: results
- A hypothesis
- More experimental evidence
- Theoretical explanation
- Summary and future work

Summary

Two competing algorithms:
- int-dual - all-integer dual simplex
  - old idea, only limited success
- cplex - industrial strength optimizer

Restricted domain (logic minimization).
Extreme differences across instances:
- time out for one algorithm vs. seconds for the other.

Want to "profile" the instances, or at least explain the differences.
A pattern?

# succ. completions, one hour
original + 32 random row/col permutations
dedicated fast processor

<table>
<thead>
<tr>
<th>alg</th>
<th>N/33</th>
<th>med.</th>
<th>mean</th>
<th>stddev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int- dual</td>
<td>29</td>
<td>15.9</td>
<td>N/a</td>
<td>N/a</td>
</tr>
<tr>
<td>cplex</td>
<td>0</td>
<td>N/a</td>
<td>N/a</td>
<td>N/a</td>
</tr>
</tbody>
</table>

Benchmark: e64.b (logic synthesis) -- a “canonical” picture

The pattern

<table>
<thead>
<tr>
<th>alg</th>
<th>N/33</th>
<th>med.</th>
<th>mean</th>
<th>stddev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int- dual</td>
<td>33</td>
<td>14.6</td>
<td>21.1</td>
<td>15.1</td>
</tr>
<tr>
<td>cplex</td>
<td>33</td>
<td>3.9</td>
<td>6.0</td>
<td>6.1</td>
</tr>
</tbody>
</table>

Benchmark: rot.b

A pattern: Not much of one

<table>
<thead>
<tr>
<th>alg</th>
<th>N/33</th>
<th>med.</th>
<th>mean</th>
<th>stddev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int- dual</td>
<td>0</td>
<td>N/a</td>
<td>N/a</td>
<td>N/a</td>
</tr>
<tr>
<td>cplex</td>
<td>33</td>
<td>3.6</td>
<td>3.5</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Benchmark: max1024

A pattern: Less pronounced

Benchmark: e64.b

Benchmark: max1024
Conversion to block structure does not improve performance

- int-dual performance: not necessarily improved
- underlying block structure
- random permutations
- better aggregate performance for int-dual
- e64 as posted: 27.5
- recall that median = 15.9
- after minimization: 55.0

Basic idea

- More underlying structure => better for int-dual
- Less underlying structure => better for cplex

Why is this interesting?
- e.g., optimum for e64.b was unknown prior to int-dual

- What do we mean by structure?
- More precise (and testable) hypothesis?
From observation to controlled experiment

- Behavior “in the wild”: algorithms on industrial instances
- Behavior “in captivity”: algorithms on carefully designed instances

Instances with pure blocks

Create K copies of a small instance, arrange on diagonal, permute rows and columns randomly 32 times to obtain a class of equivalent instances.

Construction of blocks with added rows

Create 32 instances with different random choices of the nonzeros; randomly permute each one.
Blocks used in results reported here

- **st15** - instance based on Steiner triples (15 vars, 35 const)
- **maincont** - small logic synthesis instances (61 vars, 50 const)
- **f51m** - slightly larger logic synthesis instance (175 vars, 187 const)

Details and reporting

- Class of 32 random related instances
- Intel(R) Xeon(TM) CPU 3.20GHz, 2 GB memory, 2048 KB cache
- Report runtime only (seconds, time out = 600)
- Here we report median only, but note number of successful completions, if not 32

results for st15, pure blocks

<table>
<thead>
<tr>
<th>blocks</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>int-dual</td>
<td>0.1</td>
<td>0.3</td>
<td>1.0</td>
<td>4.1</td>
<td>29.8</td>
</tr>
<tr>
<td>cplex</td>
<td>0.02</td>
<td>0.9</td>
<td>&gt; 10 hours</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Outline

- Observation in a limited domain
- Initial experiments: design
- **Initial experiments: results**
- A hypothesis
- More experimental evidence
- Theoretical explanation
- Summary and future work
results for maincont, pure blocks

<table>
<thead>
<tr>
<th>blocks</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>int-dual</td>
<td>&lt;0.1</td>
<td>&lt;0.1</td>
<td>&lt;0.1</td>
<td>0.1</td>
<td>0.3</td>
<td>1.6</td>
</tr>
<tr>
<td>cplex</td>
<td>&lt;0.1</td>
<td>&lt;0.1</td>
<td>&lt;0.1</td>
<td>0.2</td>
<td>0.5</td>
<td>&gt;600</td>
</tr>
</tbody>
</table>

29/32 completions
> 600 between 24 and 28 blocks

results for f51m, pure blocks

<table>
<thead>
<tr>
<th>blocks</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>int-dual</td>
<td>0.1</td>
<td>0.4</td>
<td>2.1</td>
<td>12.4</td>
<td>65.7</td>
</tr>
<tr>
<td>cplex</td>
<td>0.3</td>
<td>59.0</td>
<td>&gt;600</td>
<td>&gt;600</td>
<td>&gt;600</td>
</tr>
</tbody>
</table>

8 blocks with increasing row factor

<table>
<thead>
<tr>
<th>row fact.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>0.1</td>
<td>0.1</td>
<td>0.2</td>
<td>0.7</td>
<td>2.7</td>
<td>&gt;600</td>
<td>&gt;600</td>
<td>&gt;600</td>
</tr>
<tr>
<td>cplex</td>
<td>0.2</td>
<td>0.3</td>
<td>0.4</td>
<td>1.4</td>
<td>9.6</td>
<td>20.9</td>
<td>71.1</td>
<td>90.0</td>
</tr>
<tr>
<td>N/32</td>
<td>31</td>
<td>23</td>
<td>31</td>
<td>31</td>
<td>30</td>
<td>27</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

mean = 59.9
stdev = 140.5
Wild distributions!

blank = neither dominates, X = neither finishes in 1 hour
I = int-dual dominates, C = cplex dominates
this is not going well

not much hope of a statistically meaningful comparison between int-dual and cplex.

too many variables: will anything hold up

not clear what to measure
domain already very limited

... 

all is not lost: let’s stick with what we know

- int-dual performs well on pure blocks
- cplex performs badly on pure blocks

| id | 0.1 | 0.2 | 0.7 | 2.7 | >600 | >600 | >600 |

- int-dual gets worse as more random nonzeros are added
  (has been observed for columns, too)

Outline

- Observation in a limited domain
- Initial experiments: design
- Initial experiments: results
- **A hypothesis**
- More experimental evidence
- Theoretical explanation
- Summary and future work

A viable hypothesis

- int-dual runtime is predictable on pure-blocks instances and grows polynomially with increasing number of blocks
- cplex runtime on pure-blocks instances becomes erratic as soon as an instance-dependent threshold is reached
- int-dual runtime increases and/or becomes erratic as the number of nonzeros added to a pure-blocks instance is increased

erratic = “heavy tail” distribution (stdev >> mean >> median)
or time outs and overflows
why this is good

- not dependent on comparison between the algorithms
- not dependent on statistics
- extreme contrasts are what we observed initially -- so we need to deal with them
- appears to be true in a variety of circumstances
- even though instances are artificial, may explain why int-dual has limited use

---

Outline

- Observation in a limited domain
- Initial experiments: design
- Initial experiments: results
- A hypothesis
- **More experimental evidence**
- Theoretical explanation
- Summary and future work

---

f51m, added rows

3 blocks, increasing row factor

<table>
<thead>
<tr>
<th>row fact.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/32</td>
<td>23</td>
<td>22</td>
<td>19</td>
<td>12</td>
<td>15</td>
<td>10</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>3.8</td>
<td>72.6</td>
<td>52.6</td>
<td>210.2</td>
<td>&gt;600</td>
<td>&gt;600</td>
<td>&gt;600</td>
<td>&gt;600</td>
</tr>
<tr>
<td>cplex</td>
<td>&gt;600</td>
<td>&gt;600</td>
<td>260.2</td>
<td>70.2</td>
<td>17.4</td>
<td>11.0</td>
<td>9.3</td>
<td>6.4</td>
</tr>
</tbody>
</table>

| 1 | 9 | 23 | 31 | N/32 |

---

distributions that are **not** erratic

---

Outline

- Observation in a limited domain
- Initial experiments: design
- Initial experiments: results
- A hypothesis
- More experimental evidence
- **Theoretical explanation**
- Summary and future work
All-Integer Dual Simplex

vertex cover of a triangle

min \( x_1 + x_2 + x_3 \)

\[
\begin{array}{ccc}
-x_1 & -x_1 & -x_2 \\
1 & 1 & 0 \\
0 & -1 & 1 \\
-1 & -1 & 1 \\
1 & 0 & -1 \\
\end{array}
\]

tableau

choose constraint to satisfy

choose variable to do the trick

swap them now \( x_1 = 1, e_1 = 0 \)

a pivot step

and another pivot...

and another, but this one involves division by \(-2\)...

example (continued)

oops: solution is not integral

and we get fractions. The LP solution is

\[
\begin{align*}
x_1 &= \frac{1}{2}, \quad x_2 = \frac{1}{2}, \quad x_3 = \frac{1}{2} \\
\end{align*}
\]
Maintaining an integer tableau

<table>
<thead>
<tr>
<th>e1</th>
<th>e3</th>
<th>x3</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ct</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
</tr>
</tbody>
</table>

x1 = x3 = 1, cost = 2

<table>
<thead>
<tr>
<th>e1</th>
<th>e3</th>
<th>ct</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

add a new constraint that must be met by all integer solutions

int-dual: another look

0 in pivot row means no change in pivot column

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

0 in pivot column means no change in pivot row

<table>
<thead>
<tr>
<th>-1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

int-dual on “pure blocks”

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2</th>
<th>0</th>
<th>0</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

element of second block not changed by pivot in first block

why does int-dual work well on pure blocks?

- because each block is solved independently
- progress is made in whatever block contains the pivot

One out of three parts (of the hypothesis) explained
part 2: cplex does badly on pure blocks

cplex is a "branch and bound" algorithm
branching tree for one block

Note: this is a gross oversimplification

cplex is a "branch and bound" algorithm
branching tree for two blocks

Summary

- Some observations about algorithm behavior on industrial benchmarks
- Hypothesis specializes observations to a limited, verifiable domain
- Extensive experiments consistent with hypothesis
- Partial theoretical explanation for hypothesis
- Aside: int-dual dates to 1960’s; experiments in early 1970’s deemed it good for small instances of set cover (but impractical otherwise)

Future work

- better understanding of both algorithms to explain what’s going on
- "pure" branch and bound instead of cplex
- experiments to explore variations on variables (how to add rows, columns too, add -1 entries)
- observations in other domains (particular where there has been success with int-dual)
- other approaches to claiming one algorithm better than another where distributions are crazy
- can we learn from outliers, i.e., easy and hard instances of a class

Thanks...

- Xiao Yu Li, now at Amazon, started us down this path
- Eric Sills, NCSU High Performance Computing Center, helped with hardware and software availability

Source code for software and (eventually) instance classes and results at http://people.engr.ncsu.edu/mfms/Software